7545

Network Working Group
Request for Comments #215
NIC #7545
Categories: C.2, D.1, D.3, G.1
Updates: none
Obsoletes: none

Alex McKenzie
BBN
30 August 1971

NCP, ICP, and TELNET:
The Terminal IMP Implementation

By early December there will be six Terminal IMPs incorporated into the network, with additional Terminal IMPs scheduled for delivery at a rate of about one per month thereafter. For this reason the implementation of network protocols (and deviations from them) may be of interest to the network community. This note describes the choices made by the Terminal IMP system programmers where choices are permitted by the protocols, and documents some instances of non-compliance with protocols.

Most of the choices made during protocol implementation on the Terminal IMP were influenced strongly by storage limitations. The Terminal IMP has no bulk storage for buffering, and has only 8K of 16-bit words available for both device I/O buffers and program. The program must drive up to 64 terminals which generally will include a variety of terminal types with differing code sets and communication protocols (e.g., the IBM 2741 terminals). In addition, the Terminal IMP must include a rudimentary language processor which allows a terminal user to specify parameters affecting his network connections. Since the Terminal IMP exists only to provide access to the network for 64 terminals, it must be prepared to maintain 128 (simplex) network connections at any time; thus each word stored in the NCP tables on a per-connection basis consumes a significant portion of the Terminal IMP memory.

It should be remembered that the Terminal IMP is designed to provide access to the network for its users, not to provide service to the rest of the network. Thus the Terminal IMP does not contain programs to perform the "server" portion of the ICP; in fact, it does not have a "logger" socket.

The Terminal IMP program currently implements only the NCP, the ICP, and the TELNET protocol since these are of immediate interest to sites with Terminal IMPs. It is anticipated that portions of the data transfer protocol will be implemented in the future; the portions to be implemented are not yet clearly defined, but will probably include the infinite bit stream (first) and the "transparent" mode (later). Developments in the area of data transmission protocol will be documented in the future.

The remainder of this note describes, and attempts to justify, deviations from the official protocols and other design choices of interest. Although written in the present tense, there are some additional known instances of deviation from protocol which will be corrected in the near future.

A) Deviations from Protocols

1) The Terminal IMP does not guarantee correct response to ECO commands. If some Host A sends a control message containing ECOs to the Terminal IMP, and the message arrives at a time when

a) the Terminal IMP has a free buffer and

b) the control link from the Terminal IMP to Host A is not blocked

then the Terminal IMP will generate a correct ERP for each ECO. In all other cases the ECO commands will be discarded. (All control messages sent by the Terminal IMP begin with a NOP control command, so if Host A sends a control message consisting of 60 ECO commands, the Terminal IMP will answer (if at all) with a 121-byte message — 1 NOP and 60 ERPs.)

The reason for this method of implementation is that to guarantee correct response to ECO in all cases requires an infinite amount of storage. For example, suppose Host A sends control messages, each containing an ECO command, to Host B at the rate of one per second, but that Host A accepts messages from the network as slowly as possible (one every 39 seconds, say). Then Host B has only three choices which do not violate protocol:

a) Declare itself dead to the network (i.e., turn off its Ready line), thereby denying all its users use of the network.

b) Refuse to accept messages from the network faster than the slowest possible foreign Host (i.e., about one every 39 seconds). If Host B is a Terminal IMP, this is almost certainly slow enough to soon reach a steady state of no users.

c) Implement "infinite" storage for buffering messages.

Since it is clear that none of the "legal" solutions are possible, we have decided to do no buffering, which should (we guess) satisfy the protocol well over 99% of the time.

2) The Terminal IMP does not guarantee to issue CLS commands in response to "unsolicited" RFCs. There are currently several ways to "solicit" an RFC, as follows:

   a) A terminal user can tell the Terminal IMP to perform the ICP to the TELNET Logger at some foreign Host. This action "solicits" the RFCs defined by the ICP.

   b) A terminal user can send an RFC to any particular Host and socket he chooses. This "solicits" a matching RFC.

   c) A terminal user can set his own receive socket "wild." This action "solicits" an STR from anyone to his socket. Similarly, the user can set his send socket "wild" to "solicit" an RTS.

   If the Terminal IMP receives a "solicited" RFC it handles it in accordance with the protocol. If the Terminal IMP receives a control message containing one or more "unsolicited" RFCs it will either issue CLS commands or ignore the RFCs according to the criteria described above for answering ECOs (and for the same reasons). Further, if the Terminal IMP does issue a CLS in response to an unsolicited RFC it will not wait for a matching CLS before considering the sockets involved to be free for other use.

3) After issuing a CLS for a connection, the Terminal IMP will not wait forever for a matching CLS. There are two cases:

   a) The Terminal IMP has sent an RFC, grown tired of waiting for a matching RFC, and therefore issued a CLS

   b) The Terminal IMP has sent a CLS for an established connection (matching RFCs exchanged)

   In either of these cases the Terminal IMP will wait for a matching CLS for a "reasonable" time (probably 30 seconds to one minute) and will then "forget" the connection. After the connection is forgotten, the Terminal IMP will consider both sockets involved to be free for other use.

Because of program size and table size restrictions, the Terminal IMP assigns socket numbers to a terminal as a direct function of the physical address of the terminal. Thus (given this socket assignment scheme) the failure of some foreign Host to answer a CLS could permanently "hang" a terminal. It might be argued that the Terminal IMP could issue a RST to the offending Host, but this would also break the connections of other terminal users who might be performing useful work with that Host.

4) The Terminal IMP ignores all RET commands. The Terminal IMP cannot buffer very much input from the network to a given terminal due to core size limitations. Accordingly, the Terminal IMP allocates only one message and a very small number of bits (currently 120 bits; eventually some number in the range 8-4000, based on the terminal's speed) on each connection for which the Terminal IMP is the receiver. Given such small allocations, the Terminal IMP attempts to keep the usable bandwidth as high as possible by sending a new allocation, which brings the total allocation up to the maximum amount, each time that:

   a) one of the two buffers assigned to the terminal is empty, and

   b) the allocations are below the maxima.

   Thus, if a spontaneous RET were received, the reasonable thing for the Terminal IMP to do would be to immediately issue a new ALL. However, if a foreign Host had some reason for issuing a first spontaneous RET, it would probably issue a second RET as soon as it received the ALL. This would be likely to lead to an infinite (and very rapid) RET-ALL loop between the two machines, chewing up a considerable portion of the Terminal IMP's bandwidth. Since the Terminal IMP can't "afford" to communicate with such a Host, it ignores all RETs.

5) The Terminal IMP ignores all GVB commands. Implementation of GVB appears to require an unreasonable number of instructions and, at the moment at least, no Host appears to use the GVB command. If we were to implement GVB we would always RET all of both allocations and this doesn't seem very useful.

6) The Terminal IMP does not handle a total bit-allocation greater than 65,534 (2^{16}-2) correctly. If the bit-allocation is ever raised above 65,534 the Terminal IMP will treat the allocation as infinite. This treatment allows the Terminal IMP to store the bit allocation for each connection in a single word, and to avoid double precision addition and subtraction. Our reasons for this decision are:

   a) A saving of more than 100 words of memory which would be required for allocation tables and for double precision addition/subtraction routines.

   b) Our experience, which indicates that very few Hosts (probably one at most) ever raise their total bit allocation above 65,534 bits.

   c) Our expectation that any Host which ever raises its bit allocation above 65,534 probably would be willing to issue an infinite bit allocation if one were provided by the protocol. Once the bit allocation is greater than about 16,000, the message allocation (which the Terminal IMP handles correctly) is a more powerful method of controlling network loading of a Host system than bit allocation. We believe that Hosts which have loading problems will recognize this.

7) The Terminal IMP ignores the "32-bit number" in the ICP. When the Terminal IMP (the "user site") initiates the Initial Connection Protocol the actual procedure is to send the required RTS to the logger socket of the user-specified foreign Host and simultaneously to set the terminal user's send and receive sockets in a state where each will accept any RFC from the specified Host. The 32-bit socket number transmitted over the logger connection is ignored, and the first RTS and STR addressing the user's sockets will be accepted (and answered with matching RFCs).

   The ICP allows the foreign Host to transmit the RFCs involving Terminal IMP sockets "U+2" and "U+3" at any time after receipt of the RFC to the (foreign Host's) logger socket. In particular, the RFCs may arrive at the Terminal IMP before the 32-bit number. In the case

of a "normal" foreign Host, the first incoming RFCs for sockets U+2 and U+3 will come from the sockets indicated by the 32-bit number, so it doesn't matter if the number is ignored. In the case of a pathologic foreign Host, a potentially infinite number of "wrong" RFCs involving U+2 and U+3 may arrive at the Terminal IMP before the 32-bit number is sent. The Terminal IMP would be required to store this stream of RFCs pending arrival of the 32-bit number, then issue CLS commands for all "wrong" RFCs. However, the Terminal IMP does not have infinite storage available for this purpose (it is also doubtful that a terminal user really wants to converse with a pathologic foreign Host) so the Terminal IMP assumes that the foreign Host is "normal" and ignores the 32-bit number.

B) Other Design Choices Related to Protocol

1) The Terminal IMP ignores incoming ERR commands and does not output ERR commands.

2) The Terminal IMP assumes that incoming messages have the format and contents demanded by the relevant protocols. For example, the byte size of incoming TELNET messages is assumed to be 8. The major checks which the Terminal IMP does make are:

a) If an incoming control message has a byte count greater than 120 then it is discarded.

b) If a control command opcode greater than 13 is found during the processing of a control message then the remainder of the control message is discarded.

c) If an incoming data message has a byte count indicating that the bit allocation for the connection is exceeded (based on the assumed byte size) then the message is discarded.

3) If one control message contains several RST commands only one RRP is transmitted. If several control messages, each containing RST commands, arrive "close together" only one RRP is returned. [The actual implementation is to set a bit each time a RST is found (in "foreground") and to reset the bit when a RRP is sent (in "background").]

4) Socket numbers are preassigned based on the hardware "physical address" (in the terminal multiplexing device) of the terminal. The high order 16 bits of the socket number give the device number (in the range 0-63) and the low order 16 bits are normally 2 or 3 depending on the socket's gender (zero is also used during ICP). [We would be pleased to see socket number length reduced to 16 bits; in that case the high order 8 bits would be mapped to the device and the low order 8 bits would contain 2 or 3.]

5) During ICP, with the Terminal IMP as the user site, the Terminal IMP follows the "Listen" option rather than the "Init" option (as described at the top of page 3, NIC #7170). In other words, the Terminal IMP does not issue the RFCs involving sockets U+2 and U+3 except in response to incoming RFCs involving those sockets. In this context, we will mention that the "deadlock" mentioned in NWG-RFC #202 does not exist, since the ICP does not give the server the "Listen" option (see NIC #7170, page 2).